(4)

MASSACHUSETTS INSTITUTE OF TECHNOLOGY VLSI PUBLICATIONS

VLSI Memo No. 89-520
April 1989

DRIFS - A Data Retrieval Interface for Integrated Circuit Fabrication Systems

Michael P. Ruf

Abstract

People who operate a factory need to access data from many aspects of the fabrication environment. Many IC fabrication facilities store manufacturing data in distributed, heterogeneous database networks, meaning that data exists in several dissimilar databases and computer systems. Retrieval and integration of data can be a cumbersome task due to this configuration. The ideal solution to these problems is to standardize a data model, storing the manufacturing data in a single database. However, since such a standardization is not likely to occur in the near future, a more immediate solution is needed.

DRIFS (Data Retrieval Interface for Fabrication Systems) addresses the problems of integration and data retrieval by providing a standard query interface and data model for heterogeneous, distributed fabrication databases. The DRIFS prototype is described and illustrated in this thesis.

Acknowledgements

Submitted to the Department of Electrical Engineering and Computer Science at MIT on April 7, 1989 in partial fulfillment of the requirements for the *Degree of Master of Science* in Electrical Engineering and Computer Science. This work was supported in part by the Defense Advanced Research Projects Agency under contract number MDA972-88-K-0008.

Author Information

Ruf: Electrical Engineering and Computer Science, Room 36-667, MIT, Cambridge, MA 02139. (617) 253-7811.

DRIFS – A DATA RETRIEVAL INTERFACE FOR INTEGRATED CIRCUIT FABRICATION SYSTEMS

by

Michael P. Ruf

submitted in partial fulfillment

of the requirements for the degree of

MASTER OF SCIENCE

IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

April 7, 1989

Signature of Author ______________________________

Department of Electrical Engineering and Computer Science

April 7, 1989

Certified by ______________________________

Donald E. Troxel

Thesis Supervisor

Accepted by ______________________________

Arthur C. Smith

Chairman, Departmental Committee on Graduate Students

DRIFS – A Data Retrieval Interface for Integrated Circuit Fabrication Systems

by

Michael P. Ruf

Submitted to the Department of Electrical Engineering and Computer Science on April 7, 1989 in partial fulfillment of the requirements for the Degree of Master of Science in Electrical Engineering and Computer Science

Abstract

People who operate a factory need to access data from many aspects of the fabrication environment. Many IC fabrication facilities store manufacturing data in distributed, heterogeneous database networks, meaning that data exists in several dissimilar databases and computer systems. Retrieval and integration of data can be a cumbersome task due to this configuration. The ideal solution to these problems is to standardize a data model, storing the manufacturing data in a single database. However, since such a standardization is not likely to occur in the near future, a more immediate solution is needed.

DRIFS, (Data Retrieval Interface for Fabrication Systems, adresses the problems of integration and data retrieval by providing a standard query interface and data model for heterogeneous, distributed fabrication databases. The DRIFS prototype is described and illustrated in this thesis.

Thesis Supervisor: Donald E. Troxel

Title: Professor of Electrical Engineering and Computer Science

Acknowledgements

My thesis work has been done at the Computer Science Center of Texas Instruments in Dallas as my third assignment in the VI-A program. I appreciate the guidance and assistance of the following people at TI: Richard Hawkeye, Aditya Srivastava, Steve Walker, Tom Ekberg, and Cynthia Martin. I also acknowledge the support of Prof. Donald E. Troxel and of Michael Heytens. Discussions with Michael provided valuable help in the design of DRIFS. The data model for DRIFS structures was derived from GESTALT, a database he authored for computer aided fabrication.

Biography

Michael Ruf is a candidate for the degrees of Master of Science and Bachelor of Science in Electrical Engineering at the Massachusetts Institute of Technology. He worked at Texas Instruments as a member of the M.I.T. VI-A program and was a graduate research assistant for the Computer Aided Fabrication project at M.I.T. He is also a student member of the IEEE. His primary research interest is in the management of information systems, and in the computer aided fabrication of integrated circuits.

Contents

List of Figures

List of Tables

1 Introduction

With the growing complexity of integrated circuit processes, the increasing number of technologies and product lines, and the relentless demand for higher volumes, commercial IC fabrication is increasingly dependent on information management. Information is collected from almost every aspect of the manufacturing process. Financial, marketing, design, engineering, production, scheduling, resource management, parametric, and control information all contribute to a vast collection of **fabrication data**. To maintain high productivity, IC manufacturers must draw on technical, historical, and corporate data at all levels of the fabrication process. More importantly, this data must be *integrated* and *easily retrievable*.

The past decade has seen the growth of commercial MIS systems for fabrication data. These systems are responsible for handling information for all aspects of the fabrication cycle, from design through packaging. However, in most cases, the systems are piecemeal, covering only a particular stage or aspect of the fabrication process. For example, separate computers and software are commonly used for logic design, process design, scheduling, fabrication, and facility support. Because these systems do not share data or communicate efficiently with each other, they do not support *integration* of data at a global level. Furthermore, these large conglomerations of data are not always equipped with *efficient retrieval* interfaces. Due to the magnitude and complexity of such systems, often only highly trained individuals can access and interpret the data.

Improving the inadequacies of existing IC fabrication management systems will be a key factor in adapting to the growing information needs of factories of the future.

2 Fabrication Data in Heterogeneous Databases

Fabrication data can be categorized as shown in Figure 1 into two broad areas:

- **Operations Data.** This encompasses the data directly associated with the operation of the IC plant, including physical layout, engineering, scheduling facilities support, historical, parametric, work in progress (WIP), yield, and inventory data.
- **Financial/Planning Data.** This includes management level data such as costing, pricing, demand and market forecasting data as well as WIP, yield, and inventory data.

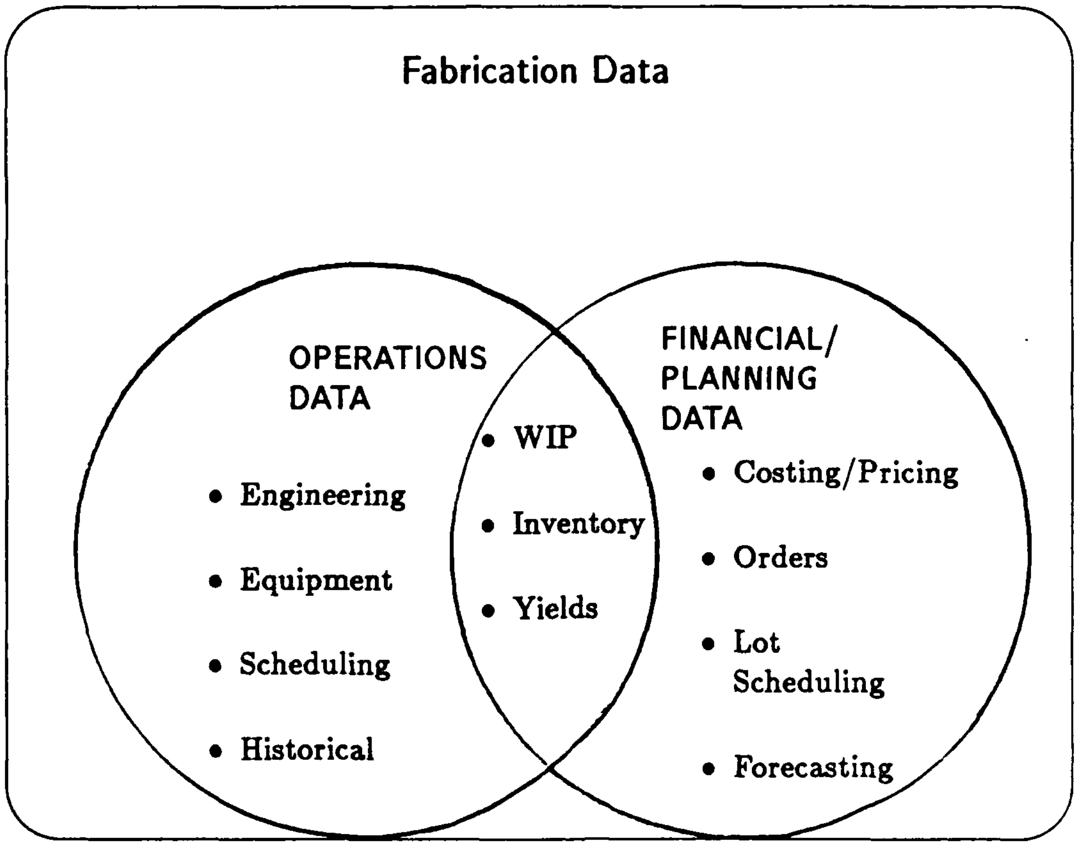


Figure 1: Broad categorization of fabrication data

2.1 Shop Floor Control Systems

The majority of the fabrication data is usually managed by a shop floor control (SFC) system. Several shop floor control (SFC) systems for IC fabrication are in use today. These systems include:

- PROMIS
- COMETS
- SMS/370
- GE-Fanuc
- CADAM

PROMIS and COMETS, the two leading commercial manufacturing systems use table-based databases. SMS/370, GE-Fanuc, and CADAM use hierarchical databases. These systems normally handle a major portion of the information needs for a particular facility. Most shop floor control systems concentrate on Operations Data, separating it into three major categories: facilities, engineering, and lot tracking/history. These categories are discussed in detail in Section 4.3.

2.2 Supplemental Data Systems

Supplemental systems are used to collect and manage additional information not covered by shop floor control systems. Such systems might specialize in the following areas:

- **Testing/yield.** Stores the results of circuit probe operations (lot and yield data). Often, summaries of this data are downloaded into the SFC databases.
- **Scheduling.** Refers to dynamic routing of lots through the facility to achieve optimum throughput, given certain constraints. Information needed for scheduling often comes from the SFC databases. However, some scheduling programs may need additional information (such as constraints) not present in the SFC database. Such scheduling systems may maintain their own databases.

- **Production/inventory planning .** Refers to information associated with planning new lot starts, desired stock levels, order commitments, etc. Production planners may draw on information from SFC and testing/yield as well as financial/marketing databases.

- **Equipment maintenance.** Includes preventive maintenance, repair, and idle time histories, mean time to repair, mean time between failure, and mean time between assists.

- **Equipment control.** Handles supplemental recipe information not covered by the SFC. This may include furnace controls and settings (recipes) associated with specific operations. alignment and sizing information for lithography equipment, etc. These systems may reside on level II or level III computers which are directly interfaced to the fabrication equipment.

- **Financial/marketing.** Handles sales order processing, standard cost accounting, market trend analysis/forecasting, etc. The detail and breadth of financial/marketing systems vary depending on the corporate structure of the plant. Many SFC systems handle financial/marketing needs, but supplemental data is often maintained on a stand-alone system, or as part or the corporate financial databases.

- **Facility support.** Monitors and controls temperatures, humidity, gas flows and pressures, liquid weights, valves, etc. Facility support systems also record air-born particle count information and particle count, resistivity, total oxidation and oxided carbon content of the deionized water used in the facility. These systems have alarms giving notification of processes that are out of control, and procedures to correct such situations. Such systems are normally stand-alone and run on level I or level II computers.

The shop floor databases and the databases which store additional manufacturing data will be referred to as *local databases*. Often, users want to perform queries which draw on data from two or more of these local databases. How these queries are executed is an integration issue.

2.3 Downloading from Supplemental Da*abases to the SFC Database

One common method of integrating the data is to download information from supplemental databases into the main shop floor control database. This method has a number of disadvantages:

- Any outside information integrated into the SFC system must conform to the structure of the SFC database. If this structure cannot accomodate a certain data model, part of the data may be forfeited in converting to a suitable model.
- Integration is not real-time. An intermediate program must port the data from supplemental database to an SFC database. Queries may generate outdated information because the intermediate program can only be run periodically.
- As applications change, or new applications are written, integration needs may change. To restructure the integration model, the schema of the SFC databases must be modified. This is a high overhead procedure. Therefore, this integration method is not dynamic with regard to information needs.
- Because all access to integrated information must be through the SFC system, the demands on the system increase as more information from other databases is duplicated in the SFC databases to support integration.

2.4 Categorizing Data Retrieval Needs

Because different people associated with the fabrication process may want access to different types of information, it is helpful to categorize these people into several groups with similar information needs. A typical breakdown is:

- **Equipment operators** run the factory machinery. They transfer lots from point to point, and initialize machines with the proper parameters for each operation.

- **Engineers.** Some engineering groups do the experimentation and groundwork for developing new processes. A process is a sequence of operations on a wafer which change its state. Processes include, wafer cleaning, deposition, exposure, development, etching, ion implantation, etc. Other engineering groups refine and sustain the processes. They work with logic and wafer profile specifications to develop new devices. They adjust processes to fit the needs of particular devices.

- **Manufacturing managers and supervisors** are responsible for the day to day activities of the factory. They monitor work in progress, handle production scheduling within the facility, identify bottlenecks, work to decrease cycle time and increase throughput, and, in general, provide for smooth operation of the facility.

- **Quality control personnel** are responsible for monitoring processes to detect abnormalities (processes gone out of control), diagnosing those abnormalities, and correcting errant processes.

- **Production planners** decide which and how many devices will be fabricated and when they will be started. They may also handle the accounting and financial affairs associated with product manufacturing.

- **Support and maintenance personnel** are responsible for maintaining and repairing equipment, and identifying and evaluating replacement needs.

- **Top level managers** provide long term direction and strategies for the facility. They are interested in summary-type data.

Most shop floor control systems provide reports tailored for certain groups listed above, but cannot provide each group with a separate "view" of the underlying schema. Hence, new reports are difficult to generate, since retrieval interfaces are not designed to address the specilized interests of each group.

3 Overview of DRIFS

DRIFS is intended to relieve the integration and ease of access problems by providing a standard query language and data model for heterogeneous, distributed fabrication databases. It is designed to provide an integrated data representation without requiring changes to the existing local databases.

DRIFS provides a software environment on a separate computer system, defining and storing a representation of each fabrication database. Using those representations, it generates queries particular to each fabrication database and retrieves data from them via computer networks.

3.1 DRIFS Schema Levels

In order to separate the tasks of providing homogeneity and integration, DRIFS breaks data acquisition into three levels: the local database level, the primitive level, and the user level.

3.1.1 Local Database Level.

The local database level consists of existing data stored in local fabrication databases. It consists of the shop floor control system database and supplemental databases, all of which are external to DRIFS.

3.1.2 DRIFS Primitive Level.

The primitive level is the crux of DRIFS, interfacing with the local database level through various networking techniques. It stores a mapping of each local database schema to the DRIFS data model. The primitive level provides homogeneity, because it interfaces to each data model of the local databases. Providing homogeneity is the most involved task of DRIFS, since many issues must be considered, such as computer networking, data models and query languages at the local level, and data formats of output files at the local level.

While data representations from separate local databases are all stored using the DRIFS data model, they are not integrated at this level.

3.1.3 DRIFS User Level.

Once the data has been mapped to the homogeneous data model of the primitive level, it can be integrated at the user level. This level combines representations from the primitive level to allow data from various local databases to be integrated in common structures. User level data representations can be tailored for specific groups such as planners or engineers, allowing each group to have separate "views" into the primitive level.

3.2 DRIFS Primitive Structures

The DRIFS data model is comprised of *primitive structures*, which are templates for data retrieved from the local databases. Each structure consists of several titled slots where the data is stored. Each slot is defined with the following fields:

- **Slot Name.** Describes what kind of data is stored in the slot.
- **Slot Type.** Specifies what format the data is stored in (i.e. string, number).
- **List/Singleton.** Tells whether the data is stored as a list of values or as a single value.

Associated with each primitive structure is information specifying from which local database to retrieve its data and how to query that database. This information indicates in which relation, file, record, etc. the data is stored. It also defines a variable map linking each slot in the primitive structure to a particular field in the local database. Primitive structures are grouped by local database, and the collection of all primitive structures for a particular local database makes up DRIFS's representation for that database.

3.3 DRIFS User Level Structures

The user level provides the means for integrating related data from each local database. User level structures combine data from the primitive structures by incorporating relevant slots from each primitive structure. The primitive structures which comprise a particular user level structure must have at least one slot in common. These key slots are used to combine the results of primitive level queries.

4 DRIFS Testbed

The Texas Instruments Dallas Logic II (DLOG-II) IC fabrication facility was used as a testbed for a prototype implementation of DRIFS. DLOG-II is a low volume, fast turn-around manufacturing facility. A significant amount of manufacturing is dedicated to special work requests. These are normally process development experiments handled as split lots, where half the lot is fabricated and acts as the control, and the other half is fabricated with certain experimental processes. Because of the developmental nature of lot fabrication at DLOG-II, the make-up of the fabrication data has emphasis on engineering and parametric data rather than financial and planning data.

4.1 DLOG-II Management Structure

DLOG-II management can be broken down into the following divisions:

- **Manufacturing.** Responsible for assuring the smooth operation of the factory. Schedules, routes, and monitors lots as they progress through the factory. Works to avoid bottlenecks and reduce cycle time.
- **Product Engineering.** Developes new processes and fabrication techniques.
- **Process Engineering.** Adjusts and refines processes and techniques to fit the needs of a particular device.
- **Systems.** Provides technical and software support and maintains the systems that manage manufacturing data.
- **Quality Control.** Does statistical analysis on process data, detecting and diagnosing abnormalities.
- **Planning.** Responsible for production planning of new parts. Calculates flow costs and cycle times associated with product manufacturing.

4.2 Manufacturing Data

The main computing resource at DLOG-II is a VAX 8650 running at 6 MIPS with 128 MB of memory. It manages the following databases:

- **PROMIS.** This is the shop floor control database. By far the largest and most involved system, demanding a major portion of the computing time.
- **Engineering Test Database.** This is a supplemental database storing test results from various integrated circuit test equipment.
- **Engineering Data Collection Database.** Stores suppl emental engineering information from PROMIS. This data is used for generating charts and trend analysis.

Additionally, DLOG-II maintains the following supplemental databases which run on separate computer systems:

- **SCADA Database.** A facility support/control system to monitor and control temperatures, humidity, gas flows and pressures, liquid weights, valves, etc. The system runs on a TI-990 mainframe computer.
- **LASS Database.** Stores alignment and sizing information by level, device, and lot. The information in this database is used in controlling lithography equipment. LASS is a stand-alone system on a micro-VAX.
- **BTU Database.** Stores furnace controls and settings (recipes) associated with specific operations. These recipes store information not covered by PROMIS. They are used to program the furnaces for a specific set of operations. This database is stored on a PDP-ll minicomputer.
- **Royco Database.** Monitors and records air-born paricle count information. Runs on a VAX minicomputer.
- **DI Water Database.** Monitors and records particle count and resistivity of the deionized water used in the facility. Runs on a VAX minicomputer.

- **TOC Database.** Monitors and records total oxidation content and oxidized carbon content of the deionized water. Runs on a VAX minicomputer.

Representations of the DLOG-II PROMIS Database and the Engineering Test Database were defined in DRIFS. Since DLOG-II does not maintain a financial database, a mock one was created with Ingres, a relational database management system. The financial database was created on a VAX 785 running ULTRIX, DEC's version of UNIX. A separate computer and operating system were chosen for the financial database to demonstrate how DRIFS would perform in a heterogeneous hardware and operating system environment.

4.3 Structure and Content of PROMIS Database

PROMIS collects and provides information for engineering, manufacturing, maintenance, and management. Most of the data it manages can be divided into three categories.

- Facility Information
- Engineering Data
- Lot Tracking and History

4.3.1 Facility Information

The facility information is defined by 24 tables. Each of these tables is stored as one record of a single ISAM file. The tables can be broken into two categories:

1. STRUCTURE
    - production areas
    - work centers
    - equipment types within work centers

- equipment units
- automated equipment
- inventory locations

2. CHARACTERISTICS and OPERATION:

   - equipment statuses
   - lot types
   - material types
   - reject information
   - rework information
   - product stages
   - labor types
   - order statuses

Off line modification of the tables is supported through menu functions, which take "snapshots" of on line tables. The user may edit the copies, check their consistency, and install final versions. The information in the tables may be retrieved through specialized reports actived from the PROMIS menu, or it may be extracted through a general file extraction facility.

The 24 tables are stored in an ISAM file named TBLS. This file contains one record for each entry of each data table in the system. The record format as a key value followed by a fixed length byte buffer. Separate table access routines can then reference the byte buffer as a set of fields. The maximum length of a character field is 130 characters. Certain fields in some files are stored as composite fields, composed of information from two other fields, concatenated together. Composite may not be entered directly, but are automatically generated by PROMIS. The following provides a brief description of each record of the PROMIS TBLS file.

- **AUTO record.** Stores information pertaining to automated equipment. There is one entry for each equipment unit. The data in this table supplements that in the EQUN record.

- **ECAT record.** Describes characteristics of automated equipment that is common to certain groups of units.
- **EQST record.** Stores codes which are used to describe the statuses of equipment units. Associated with each status are descriptive codes which indicate the equipment's availability for lot trackin.
- **EQUN record.** Stores information describing individual units of equipment in the facility. It is used by lot tracking, equipment reporting, etc.
- **ESET record.** Outdated.
- **GTNK record.** Stores information pertaining to Gas Tanks.
- **INVT record.** Describes characteristics of inventory locations in the Facility
- **LABT record.** Stores labor types for labor capacity requirements in the planning system.
- **LOTT record.** Defines lot type codes, which indicate ownership and tracking aspects of lots.
- **MATT record.** Records material types and type codes. Material types are used to generate certain prompts in lot tracking.
- **ORDS record.** Defines order status codes used by the Sales Order Entry and Planning Systems.
- **PRAR record.** Describes production areas, the highest level of description in the hierarchical PROMIS site tables.
- **PRSY record.** Describes PROMIS systems in a multi-system (multi-site) configuration.
- **PSTA record.** Defines stages, which are groups of one or more sequential process steps.
- **REJC record.** Defines reject categories for wafers, die, finished chips, and certain material types. Used by the lot tracking system.
- **REPD record.** Contains information regarding report directories and access privileges.

- **REPT record.** Stores the title used in production report headers.
- **REWK record.** Defines rework codes used in lot tracking during rework and recovery.
- **ROUT record.** Used by DLOG-II to define equipment types, which are groups of equipment units with similar characteristics.
- **TRMP record.** Outdated.
- **UNIT record.** Describes base units of measurement, including SECS-specified and user-defined units.
- **UTYP record.** Contains information to allow interconversion of certain units.
- **WAFT record.** Defines wafer types used in lot tracking.
- **WRKC record.** Describes work centers.

The PROMIS tables containing the most extensive and significant information are those describing the production areas, work centers, equipment types and equipment units. These tables are hierarchically related. Production areas are groups of work centers. Work centers and equipment types are groups of equipment units.

Production Areas The production area is the highest hierarchical level of information pertaining to a site. Examples of production areas include:

- **Front End.** This is the main production area. Blank silicon wafers enter a front end, and processed wafers leave in sealed containers to be packaged.
- **Back End.** Here, the processed wafers are cut and packaged in ceramic casings. The final products are sample-tested.
- **Reticle Fab.** This is where new masks for lithography are made. Although the reticle fab may be entered as a separate production area, reticles are normally produced in the front end or ordered from outside suppliers.

Production areas are described in the PRAR record.

Work Centers A work center is an area of the facility where a particular type of operation is performed on a lot. Examples of workcenters include:

- Wafer Cleaning
- Deposition
- Furnace
- Implant
- Stepper
- Plasma Etch
- Metallization
- Test Probe
- Failure Analysis/Repair

Each work center has a field indicating its parent production area.

Equipment Types Equipment Types are groups of equipment units which perform similar functions. For example, a furnace work center of a large front end might contain as many as 30 or 40 furnace tubes of varying functions: initial oxidation, CMOS oxidation, gate oxidation, LPCVD nitride, etc. Each furnace tube could entered as an equipment unit and categorized into separate equipment types. All of these equipment types would belong to the furnace work center.

Equipment Units Equipment Units identify individual pieces of equipment in the factory. They represent the lowest hierarchical level of information describing the site. Equipment units may or may not be available for lot track-in. For example, a positive resist developer may have lots regularly tracked in and out, but an arsine/phosphine alarm would not. Both could be entered as equipment units. The status of equipment units are modified in a separate on line file.

4.3.2 Engineering Data

Most of the engineering data in PROMIS is associated with IC process specification. PROMIS records processing instructions defined by engineers and relays it to operators or automated fabrication equipment. PROMIS also collects engineering data such as measurements and test results from operators or automated equipment. This data is used by management, quality control, and engineering for yield analysis, engineering analysis, quality assurance, etc. Promis breaks down process specification into four levels of detail:

- Device
- Process
- Recipe
- Operation

The device is at the top of the hierarchy, providing the most general information. Each of the next levels incorporates a greater amount of detail on how to manufacture the IC device. The operation is at the bottom of the hierarchy, giving the highest level of detail. Once all the necessary modules have been defined, they are combined to from a complete manufacturing process flow.

Devices Device records most commonly describe summary level instructions for processing wafers to manufacture IC's. However, they can also be used to describe other types of parts, both manufactured and purchased. For example, device records are used to describe reticles used in the facility. Since most of these reticles are purchased from photolabs, device reticle device records don't normally contain any manufacturing information. When unprocessed wafers come into the factory, they undergo a certain amount of epitaxial pre-processing, and are stored in inventory as "raw" wafers. An inventory may stock various types of raw wafers, depending on the type of epitaxial pre-processing prescribed by different IC's. Raw wafers are entered as devices and are referenced as starting material for other devices. A device's starting material is not limited to raw wafers. In some cases, the same initial process flow can be used manufacture a number of different devices. In this situation, a device is created to define the initial process flow for the partially processed wafers which will serve as starting materials for the finished devices. For example, initial wafers for gate arrays can be manufactured in mass and later customized for with additional processing particular devices.

The information in a device record can be divided into 3 major categories:

- **Administrative.** Contains descriptive information such as dates, the device record's history, personnel associated with the device, categories, etc.
- **Parametric.** Contains parameter names and values. Parameters may specify such things as lithography masks, probe tests, equipment settings, etc. They allow a general description of a device type to be tailored for specific devices.
- **Instructional.** Contains a description of the manufacturing flow for the device. Instructions can specify how to start making a device, specify a process flow, describe inventories for the device, or describe a "nested" device. All devices have an instruction of the first type, and most have a final inventory instruction. Some devices, such as purchased parts, do not have process instructions.

Processes PROMIS process records provide a higher level of detail than the devices which reference them. Several different devices may reference the same process. In this situation, each device applies its own set of devices parameters to the process. Different sets of processes are defined for each type of IC (e.g. CMOS, NMOS, PMOS, BIPOLAR). Process information is divided into two major categories: administrative and instructional. Each record contains administrative information such as dates, planning information, statuses, etc. as well as a sequence of instructions, or steps. Processes typically have between 50 and 100 steps, each one naming a recipe to be used at that point. A single process may use the same recipe at different process steps. In addition to recipe names, each process step may specify optional data, such as rework information and control wafer instructions.

Recipes A recipe describes a sequence of operations on a lot at a particular work center on a particular equipment type. Each step, or operation of a recipe describes a set of actions to be performed by equipment operators. A recipe may call the same operation at different steps, and the same operation may be called by several different recipes. Lots are tracked at the recipe level. Therefore, each recipe record contains tracking information, such as data collection operations, facility monitor (FM) points, and prerequisite FM values for lot mark-in. Recipes also contain general information such as dates, statuses, equipment type, processing times, equipment set up, equipment capacity and load sizes, input parts, etc.

Operations Operations are the most detailed building blocks for processing specifications. Each operation describes the processing instructions at a single step in a recipe. There are four types of operations:

- Fabrication Operations
- Identify Operations
- Data Collection Operations
- Hybrid Operations

Fabrication operations are textual instructions which are displayed on an operator's terminal at the lot's current work center. Since only a single line of text is provided, instructions are usually highly abbreviated.

Identify operations instruct the operator to scribe a system-generated serial number on each unit (wafer) in the lot. After an identify operation, units in a lot can be referenced individually by their serial numbers.

Data collection operations specify certain data to be collected regarding individual die, wafers, a lot as a whole, or a piece of equipment. Instructions on how to collect the data are provided in as a single line of text.

Hybrid operations are used to assemble a single part from two separate parts.

4.3.3 Lot Data

PROMIS continually collects information on each lot as it is processed. Lot data is stored in two files: the active lot file and the lot history file. The active lot file contains one record for each lot, describing what is currently happening to the lot. Active lot records are updated at each work center. The lot history file stores an account of what happened to a lot at each step of processing. Each time a lot is tracked into a new work center, lot history entries are recorded. Therefore, each lot may have many lot history records. Lot data is grouped into four categories:

- Lot Info
- Step Info
- Data Collection Info
- Wafer Details

Lot info specifies general lot level information such as lot ID, device name, lot level yield, start and current sizes, dates, and planning information. Step info includes work center, recipe, step, operator, start time, end time, start size, end size, step yield, and control, rework, hold, and reject data. Data collection info includes all the data collected at each step of lot processing. Wafer details specify information for each wafer in the lot, including wafer ID's, wafer state, and rework information.

4.4 Structure and Content of Engineering Test Database

The engineering test database (TDB) was designed and implemented by DLOG-II to manage control test data from integrated circuit test equipment. Each kind of test equipment at DLOG-II produces a different type and format of data. In fact, data formats can change from session to session on the same tester. The engineering test database consolidates the floating format test data from all the machines, and provides a standard query mechanism to the data.

The TDB is stored in VAX datalog format. Datalog files are composed of sequential ASCII variable length text records. There is one datalog file for each test session performed on each lot by each tester. The ASCII datalog files contain three types of records:

- Header Records
- Format Records
- Data Records

There are two types of data records: test session data records, and test value data records. Test session data records contain the information necessary to locate a set of test data for any given wafer. Test value data records contain the test results for a given test session. A single test session data record precedes each set of test value data records.

Header records define a format for the test session records. Test header records, and therefore test session data records, must specify the following mandatory fields:

- TESTER
- TECH (technology)
- DEVICE
- LOT
- WAFER (slice number)
- TEMP (temperature)
- STATUS
- TEST-DATE
- TEST-TIME

Since only one lot is allowed per datalog file, the TESTER, TECH, DEVICE, and LOT fields will remain constant throughout the file.

```
HDR1 TESTER TECH DEVICE LOT WAFER TEMP STATUS TEST-DATE TEST-TIME
FMT2 SUPPLY
FMT3 ICC2 ICC3 ICC5 VIL VIH VOH VOL
DAT1 SENTRY50 DMOS DPU-1 137380 23 25C PREBURN_IN 861231 14:49:34
DAT2 LHH
DAT3 832.0E-03 -2.000E-03 174.14E-03 1.399 1.99 1.988 338.1E-03
DAT3 821.0E-03 -2.000E-03 175.14E-03 1.320 1.98 1.981 340.2E-03
⋮
```

Figure 2: Example of a TDB file.

Format records define the format for test value data records. They specify field names which describe the values in the data records that follow. Figure 2 is an example of a datalog TDB file with header, format, test session data, and test value data records.

The datalog files composing the test database are hierarchically categorized by tester, technology, device and lot stored separate VMS directories. The TDB stores data from the following testers:

- **VTPROBE.** An in line tester used early in the fabrication process. Measures threshold voltages.
- **BECOLT.** A back end tester used after packaging.
- **FECOLT.** A front end tester used in laser repair.
- **SENTRY50.** A back end tester similar to BECOLT.
- **KIETHLY.** A parametric tester. Measures voltages, resistance, and currents on test bars.

4.5 Structure and Content of Financial Database

Because DLOG-II does not maintain a networked financial database, a small database was created using one Ingres relation. The relation stores flow cost and yield data associated with each device. Actual cost data was taken from a separate shop floor control system (SMS-370) at another TI facility. This data was randomly modified so as not to expose confidential information. The relation used to store the data is composed of the following domains:

- **devid.** Device ID.
- **nobars.** Number of bars, or die, per slice.
- **cqflowcost.** Average flow cost per slice for the current quarter.
- **cqyld.** Average yield for the current quarter.
- **pqflowcost.** Average flow cost per slice for the previous quarter.
- **pqyld.** Average yield for the previous quarter.

The the values for the devid field were chosen to match the DEVNAME field (a composite key field) from selected devices in the PROMIS database. The values for the remaining fields were supplied by the modified cost data. Figure 3 is an Ingres-generated listing of the of the financial database.

4.6 Database Retrieval Mechanism for PROMIS

PROMIS has a module, DATALINK, which provides the ability to extract and manipulate data from any PROMIS file. The extraction procedure can only be activated via the DATALINK menu in the PROMIS system. Once selected, this menu provides general extraction functions as well as data conversion functions. The General extract function provides an interactive method of retrieving data from individual PROMIS files. The extracted data is stored in an ascii work file in the user's PROMIS directory. After extraction, the user can select DATALINK commands to view the extracted data.

```
cost relation

|devid          |nobars|cqflowcost|cqyld |pqflowcost|pqyld |
|------------------------------------------------------------|
|7202AEP1-01.01 |   504|   286.060|    63|   252.770|    56|
|8847C2-02.01   |  3608|   358.440|    54|   372.000|    51|
|CF94553C-04.01 |  2304|   260.050|    41|   224.900|    40|
|BULL-A01.02    |  7591|   299.800|    66|   298.130|    73|
|CIMTDI-01.04   |  1277|   335.150|    82|   265.850|    66|
|58225D-01.09   |  1903|   265.960|    66|   225.840|    61|
|8847C2-02.01   |  2460|   234.110|    69|   210.880|    69|
|9000CS-15.02   |   645|   336.020|    64|   328.140|    57|
|ACT8836A-06.03 |  2070|   257.000|    58|   377.000|    55|
|ACT8867-02.02  |  1822|   250.420|    65|   203.630|    65|
|ACT8837-06.03  |  2602|   406.000|    49|   337.000|    44|
|ACT8842-01.07  |  1691|   260.690|    62|   232.010|    57|
|9100A-02.05    |  1224|   257.350|    56|   250.740|    52|
|ACS-FASTR3.01  |  6012|   369.830|    84|   358.000|    68|
|9989D-16.01    |  3474|   241.370|    87|   188.330|    68|
|ACT2150G-BP3.01|  7379|   244.210|    90|     0.000|     0|
|ACT2158-02.02  |  2704|   250.730|    86|     0.000|     0|
|ACT2158T-02.01 |   864|   308.160|    42|   254.640|    39|
|ACT2160-01.01  |   551|   297.010|    53|   294.710|    53|
|ACT2702-01.02  |   153|   334.480|    30|     0.000|     0|
|ACT7202A-03.01 |   332|   261.870|    33|   239.570|    24|
|ACT7204-03.02  |   360|   228.670|    43|     0.000|     0|
|ACTGCX5-05.01  |  1399|   227.200|    42|   224.610|    34|
|ALICAT-04.02   |   975|   228.830|    39|   179.600|    35|
|BIC256F-W01.02 |  1899|   204.640|    58|   222.350|    53|
|ASAM-04.01     |  1664|   261.910|    88|   247.550|    88|
|ASAM-TS.02     |  1676|   236.920|    84|   239.630|    84|
|BRAHMA-07.04   |  7002|   180.340|    93|   172.600|    85|
|------------------------------------------------------------|
```

Figure 3: Listing of financial database.

Column Display

```
(Scalar)  (Scalar)      (Array)
LOTID     ENGINEER      WAFNUMS
-------------------------------
10245.1   SMITH         .21
10245.1   SMITH         .22
10245.1   SMITH         .23
```

Row Display

```
LOTID     ENGINEER      WAFNUM_1    WAFNUM_2    WAFNUM_3
--------------------------------------------------------
10245.1   SMITH         .21         .22         .23
```

Figure 4: Column and row formats in PROMIS

LOTID and ENGINEER are scalars and WAFNUM is an array.

After initiating the General extract function, the user must specify which file is to be queried, name the work file, and the extraction and search criteria. The extraction criteria identify which fields are to be extracted from each entry, and the search criteria, or query constraints, specify which entries to extract. When selecting fields for extraction or search criteria, PROMIS provides on line help menus which describe every file, record and field in the PROMIS database. PROMIS records contain two types of fields: Scalar and Array. When array fields are selected, for extraction, PROMIS allows either a row or a column format. If row format is selected, only one line of data is extracted per entry, with the array field expanded within the line. If column format is selected a separate line of is used for each ***element*** in the array, with the scalar data in the entry duplicated for each line. If row output is chosen, the user must indicate how many array elements are to be displayed in the work file. The selection of row/column format is important, because it specifies not only how the data is displayed on the screen, but also how it is stored in the work file. Figure 4 shows column and row display for the same data.

Search criteria are entered as a field name, a relational operator, and a value. The following relational operators are allowed:

- EQ or =
- NE or /=
- GT or >
- LT or <
- GE or >=
- LE or <=
- IN or [
- FR

IN allows up to 10 values to be entered individually. FR allows values to be read from a work file. When more than one search criterion is entered, PROMIS extracts only the entries which meet all the criteria (i.e. the intersection of entries matching each criterion). Once the extraction is complete, PROMIS can convert the work file to several different formats, including the standard data interchange format (DIF).

4.7 Database Retrieval Mechanism for the Engineering Test Database

The engineering test database provides a program, TDBEXTR, to extract data. The extract criteria, or query constraints, can be entered from the VMS command line. TDBEXTR generates three output files: a description file, a data file, and a log file. The data and description files are in Enhansys format. Enhansys is a set of software tools for analyzing and manipulating engineering data.

The extract criteria are specified using a simple extract language. Each criterion begins with a field name and is followed by a list of values for that field, enclosed in parenthesis. The values can be singular, a list separated by commas, a range separated by two periods, or a wild card expression. If more than one extraction criterion is entered, TDBEXTR only returns the entries which match all of the criteria.

4.8 Database Retrieval Mechanism for the Financial Database

Data is retrieved from the financial database using QUEL, the standard query language for Ingres. Query constraints in QUEL are called clauses. Each clause consisits of a pair of expressions separated by a comparison operator. Comparison operators can be any of the following:

- <
- <=
- >
- >=
- =
- ! =

Expressions can be any of the following:

- Constant. (string, integer, or floating point)
- Attribute.
- Functional expression. [sin(n), cos(n), exp(n), etc.]
- Aggregate or aggregate function. (sum, count, avg, etc.)
- Combination of numeric expressions and arithmetic operators.

Basic queries can be formulated using only constant and attribute expressions. Attributes take the following form:

variable.domain

Variable specifies a particular relation, and *domain* identifies a field in that relation. A typical clause might consist of an attribute followed by a comparison operator followed by a constant. For example:

cost.nobars < 200

Here *cost* identifies the Cost Relation and *nobars* is a field, or domain, in that relation. Before this clause can be evaluated, the *cost* variable must be linked to the Cost Relation in a separate QUEL command. QUEL clauses can be linked together with logical operators to form a qualification. The following logical operators are allowed: not, and, or. An Ingres retrieve command specifies the relation and domains to extract from each tuple, as well as a qualification indicating which tuples to retrieve. Normally, Ingres will print the results of the query on the screen, but an optional argument to the retrieve command can specify a new relation to hold the output data. Ingres also provides a copy command to port data from an ingres relation to an outside file.

5 DRIFS Prototype

DRIFS was prototyped on a Texas Instruments Explorer LISP machine. The Explorer is a single-user workstation designed for rapid development and prototyping in a symbolic processing environment. Running Common LISP with incremental garbage collection, the Explorer can manage up to 128 megabytes of virtual memory. The internal Nubus architecture uses a 32-bit LISP processor running at 10 megahertz (100 ns clock period). An ethernet controller is provided for communications with local area networks. Among the networking services available are:

- Transparent file I/O
- Remote login.
- Task-to-task communication.

To support these services, the Explorer uses the following communication protocols:

- Chaosnet
- TCP/IP
- NFS
- DECnet

Chaosnet is used for communications between LISP machines. TCP/IP is a standard communications protocol adopted by the Department of Defense and other users of wide-area networks. NFS is a UNIX standard for transparent file access. It is used in conjunction with the TCP/IP protocol. DECnet is used for communications with systems produced by Digital Equipment Corporation, allowing transparent file access, inter-task communication, and remote login.

Figure 5 shows the networking configuration for the Explorer used to prototype DRIFSThe networking names of the machines are shown in parenthesis.

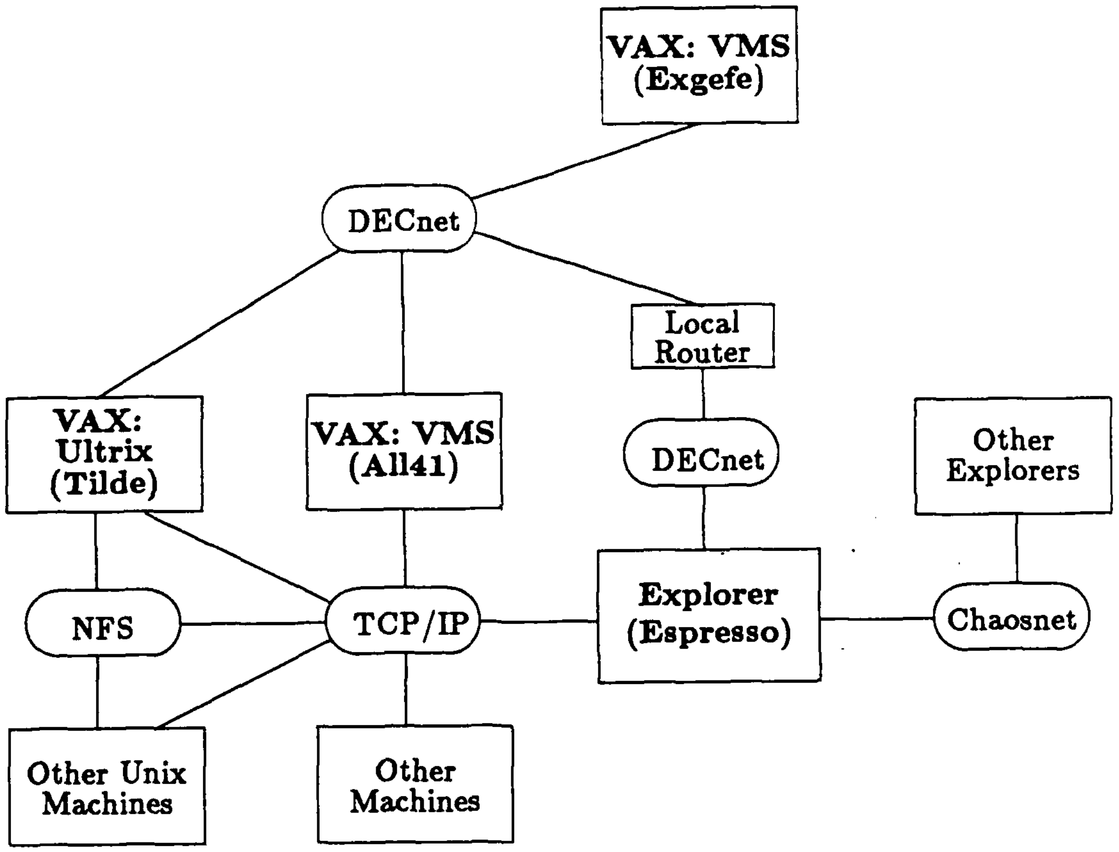


Figure 5: Partial schematic representation of Ethernet network at TI Computer Science Center

The networking name of each computer is shown in parenthesis.

5.1 DRIFS Software Environment

5.1.1 Window Interface

The DRIFS window interface (Figure 6) is broken into the following panes which allow the user to easily interact with the DRIFS environment:

- **Five command panes (bottom).** Used for defining, modifying, accessing, saving, and restoring primitive and user level structures. Also used for retrieving data and providing help features.
- **Interaction pane (upper left).** A scrollable window used to display mouse-selectable items that can be described in the Type-out pane.
- **Type-out pane (upper right).** A scrollable window used for displaying the structure descriptions, the progress of an active query, and retrieved data.

5.1.2 On Line Help Facility

The PROMIS Help command in the lower right pane allows the user to view any level of on line help available from the PROMIS menus. It is intended as an aid in creating primitive structures.

The text for this help command was automatically retrieved from PROMIS via a LISP ascii-translating character stream connected to EXGEFE. Once retrieved, the help data was parsed into LISP lists and stored on disk as help files. Each help file corresponds to a data file in PROMIS and provides information regarding its structure and fields. When requested, each file is read into a buffer that can hold up to 10 help files. The buffer management is transparent to higher level routines which access help data. When the user selects the PROMIS help command, all of the help files are displayed in the Interaction pane. He may then click on the PROMIS file of interest to see a description of that file in the Type-out pane. The Interaction pane will then display all of the names of the fields for that file. He may then select any field to see its description. This hierarchy is used for all PROMIS files with one exception: then TBLS file, which has several different records, each

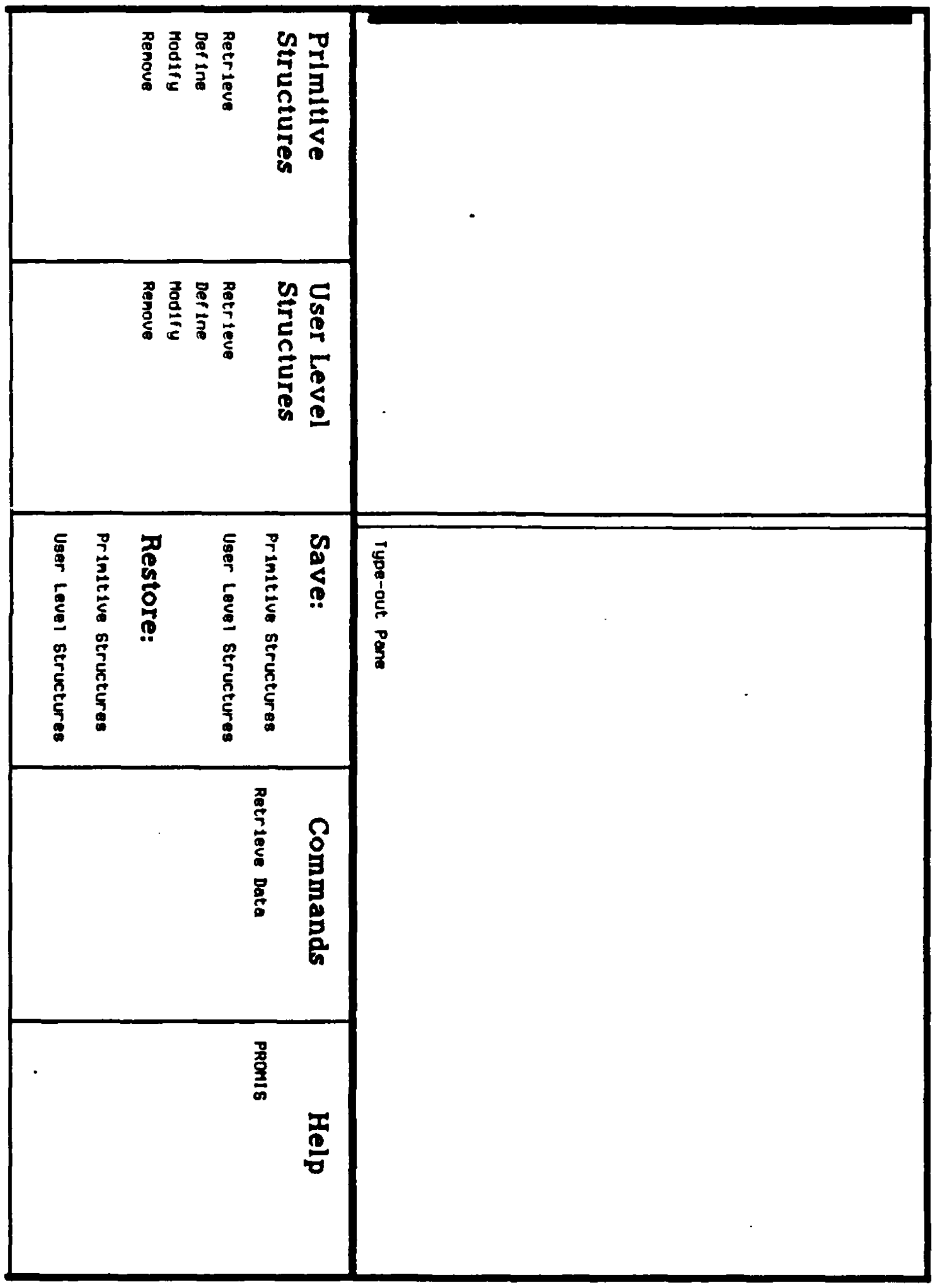


Figure 6: Drifs Window Interface

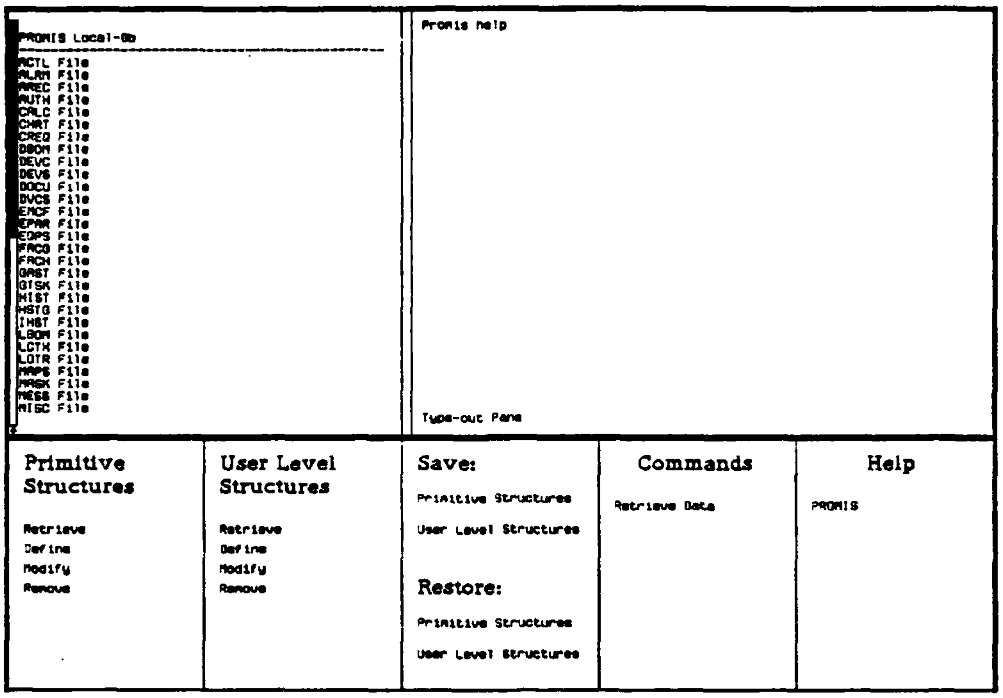
having its own fields. In this case, the user will first select the TBLS file, then a record, then any fields within that record. Figures 7- 8 show a typical progression of the PROMIS Help command.

5.1.3 Creating Primitive Structures

Primitive structures (described in Section 3.2) are created and manipulated via pop-up menus. When defining a structure, the user must specify a DRIFS structure name, a local database, a query type for that database, and each slot in the structure. Once a primitive structure is completely defined, DRIFS creates a LISP flavor as a template for data retrieved from the local databases. The flavor for a primitive structure has an instance variable for each slot in that structure. When data is retrieved from a local database it is parsed into primitive structures, and stored in flavor instances. The flavor instances are displayed in the Interaction pane, and the user may select them to view their data in the Type-out pane.

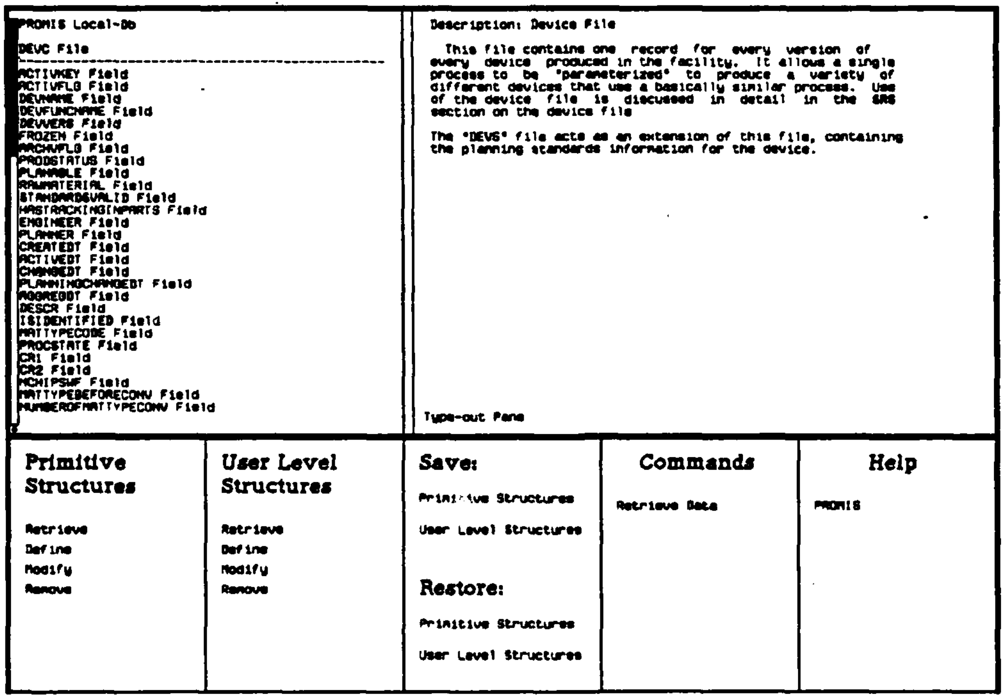
DRIFS identifies a set of query types associated with each local database: PROMIS, engineering test, and financial. Ideally, only one query type would be needed for each database. However, since the file structure for the PROMIS database is not consistent, a separate query type was defined for reading the TBLS file, which has several different record types. Slots are defined by specifying a slot name, slot type, and list/singleton specifier. The slot name is used only by DRIFS as a reference, the slot type indicates the type of data which will be stored in the slot. For most purposes, "string" and "number" should be sufficient as data types, since LISP handles parsing between numerical formats (e.g. fixnum, bignum, flonum) automatically. Finally, the list/singleton specifier indicates whether the data will be stored as a single item, or as a list of zero or more items. Figure 9 shows the DEVICE structure being defined.

Once the primitive structure has been defined, the user must create a mapping which associates each slot in the structure with a specific field in the PROMIS database file. The PROMIS fields are selected using pop-up menus, which are generated from the help files. Since PROMIS treats array fields differently from other fields, as described in Section 4.6, they must be handled specially by DRIFS. When an array field is selected in a mapping, DRIFS asks for the maximum number



```
PROMIS Local-Db
------------------------------------------------
ACTL File
ALRM File
AREC File
AUTH File
CALC File
CHRT File
CREQ File
DBOM File
DEVC File
DEVS File
DOCU File
DVCS File
EMCF File
EPAR File
EQPS File
FACG File
FACH File
GAST File
GTSK File
HIST File
HSTG File
IHST File
LBOM File
LCTX File
LOTR File
MAPS File
MASK File
MESS File
MISC File
```

```
Promis help
```

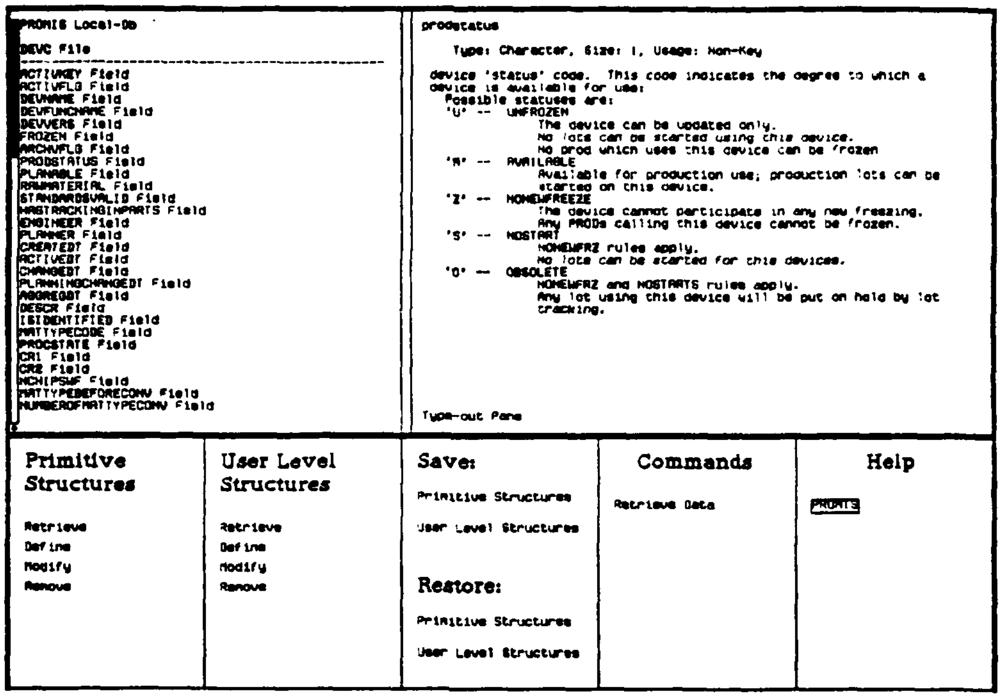


```
PROMIS Local-Db

DEVC File
------------------------------------------------
ACTIVKEY Field
ACTIVFLG Field
DEVNAME Field
DEVFUNCNAME Field
DEVVERS Field
FROZEN Field
ARCHVFLG Field
PRODSTATUS Field
PLANABLE Field
RAWMATERIAL Field
STANDARDSVALID Field
HASTRACKINGINPARTS Field
ENGINEER Field
PLANNER Field
CREATEDT Field
ACTIVEDT Field
CHANGEDT Field
PLANNINGCHANGEDT Field
AGGREGDT Field
DESCR Field
ISIDENTIFIED Field
MATTYPECODE Field
PROCSTATE Field
CR1 Field
CR2 Field
NCHIPSWF Field
MATTYPEBEFORECONV Field
NUMBEROFMATTYPECONV Field
```

```
Description: Device File

  This file contains one  record  for  every  version  of
every  device  produced in the facility.  It allows a single
process to  be  "parameterized"  to  produce  a  variety  of
different devices that use a basically similar process.  Use
of the device  file  is  discussed  in  detail  in  the  SRS
section on the device file

The "DEVS" file acts as an extension of this file, containing
the planning standards information for the device.
```

Figure 7: Promis Help command, file level.

The DEVC file was selected.

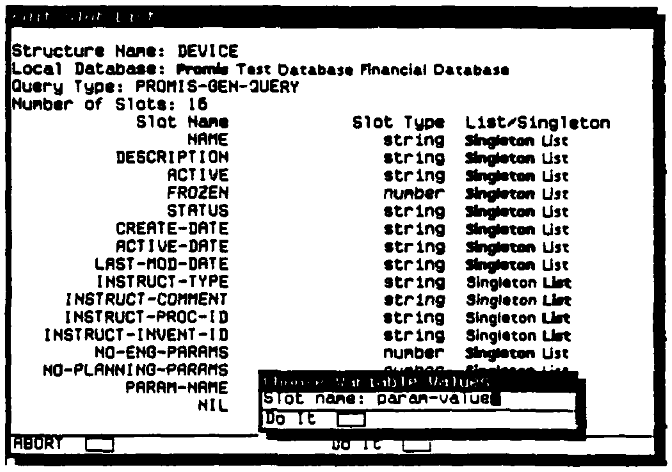


```
PROMIS Local-Db

DEVC File
--------------------------------------------------
ACTIVKEY Field
ACTIVFLG Field
DEVNAME Field
DEVFUNCNAME Field
DEVVERS Field
FROZEN Field
ARCHVFLG Field
PRODSTATUS Field
PLANABLE Field
RAWMATERIAL Field
STANDARDSVALID Field
HASTRACKINGINPARTS Field
ENGINEER Field
PLANNER Field
CREATEDT Field
ACTIVEDT Field
CHANGEDT Field
PLANNINGCHANGEDT Field
AGGREGDT Field
DESCR Field
ISIDENTIFIED Field
MATTYPECODE Field
PROCSTATE Field
CR1 Field
CR2 Field
NCHIPSWF Field
MATTYPEBEFORECONV Field
NUMBEROFMATTYPECONV Field
```

```
prodstatus

    Type: Character, Size: 1, Usage: Non-Key

 device 'status' code.  This code indicates the degree to which a
 device is available for use:
   Possible statuses are:
   'U' --  UNFROZEN
                The device can be updated only.
                No lots can be started using this device.
                No prod which uses this device can be frozen
   'A' --  AVAILABLE
                Available for production use; production lots can be
                started on this device.
   'Z' --  NONEWFREEZE
                The device cannot participate in any new freezing.
                Any PRODs calling this device cannot be frozen.
   'S' --  NOSTART
                NONEWFRZ rules apply.
                No lots can be started for this devices.
   'O' --  OBSOLETE
                NONEWFRZ and NOSTARTS rules apply.
                Any lot using this device will be put on hold by lot
                tracking.

Type-out Pane
```

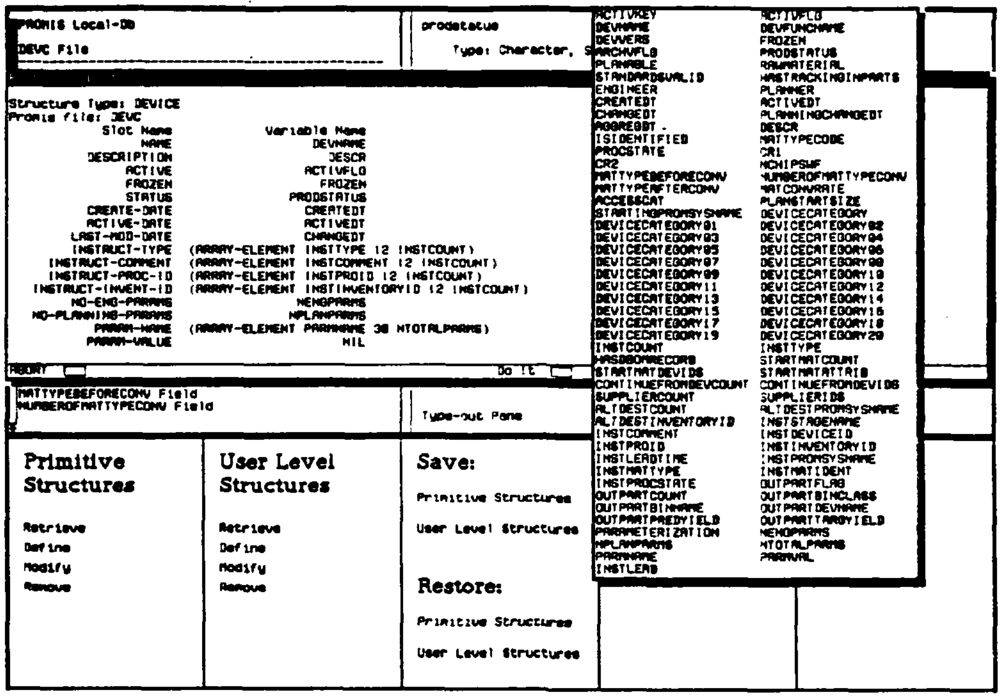
Figure 8: Promis Help command, field description.

The PRODSTATUS field was selected.



```
Structure Name: DEVICE
Local Database: Promis Test Database Financial Database
Query Type: PROMIS-GEN-QUERY
Number of Slots: 16
             Slot Name          Slot Type  List/Singleton
                  NAME             string  Singleton List
           DESCRIPTION             string  Singleton List
                ACTIVE             string  Singleton List
                FROZEN             number  Singleton List
                STATUS             string  Singleton List
           CREATE-DATE             string  Singleton List
           ACTIVE-DATE             string  Singleton List
         LAST-MOD-DATE             string  Singleton List
         INSTRUCT-TYPE             string  Singleton List
      INSTRUCT-COMMENT             string  Singleton List
      INSTRUCT-PROC-ID             string  Singleton List
    INSTRUCT-INVENT-ID             string  Singleton List
         NO-ENG-PARAMS             number  Singleton List
    NO-PLANNING-PARAMS
            PARAM-NAME
                   NIL
```

Figure 9: Sample structure definition menu.



```
PROMIS Local-Db
DEVC File

Structure Type: DEVICE
Promis file: DEVC
                  Slot Name                      Variable Name
                       NAME                            DEVNAME
                DESCRIPTION                              DESCR
                     ACTIVE                           ACTIVFLG
                     FROZEN                             FROZEN
                     STATUS                         PRODSTATUS
                CREATE-DATE                          CREATEDT
                ACTIVE-DATE                          ACTIVEDT
              LAST-MOD-DATE                          CHANGEDT
             INSTRUCT-TYPE  (ARRAY-ELEMENT INSTTYPE 12 INSTCOUNT)
          INSTRUCT-COMMENT  (ARRAY-ELEMENT INSTCOMMENT 12 INSTCOUNT)
          INSTRUCT-PROC-ID  (ARRAY-ELEMENT INSTPROID 12 INSTCOUNT)
        INSTRUCT-INVENT-ID  (ARRAY-ELEMENT INSTINVENTORYID 12 INSTCOUNT)
              NO-ENG-PARAMS                         NENGPARAMS
         NO-PLANNING-PARAMS                         NPLANPARAMS
                 PARAM-NAME (ARRAY-ELEMENT PARAMNAME 30 NTOTALPARAMS)
                PARAM-VALUE                                NIL

MATTYPEBEFORECONV Field
NUMBEROFMATTYPECONV Field

prodstatus
    Type: Character, S

ACTIVKEY                    ACTIVFLG
DEVNAME                     DEVFUNCNAME
DEVVERS                     FROZEN
ARCHVFLG                    PRODSTATUS
PLANABLE                    RAWMATERIAL
STANDARDSVALID              HASTRACKINGINPARTS
ENGINEER                    PLANNER
CREATEDT                    ACTIVEDT
CHANGEDT                    PLANNINGCHANGEDT
AGGREGDT .                  DESCR
ISIDENTIFIED                MATTYPECODE
PROCSTATE                   CR1
CR2                         NCHIPSWF
MATTYPEBEFORECONV           NUMBEROFMATTYPECONV
MATTYPEAFTERCONV            MATCONVRATE
ACCESSCAT                   PLANSTARTSIZE
STARTINGPROMSYSNAME         DEVICECATEGORY
DEVICECATEGORY01            DEVICECATEGORY02
DEVICECATEGORY03            DEVICECATEGORY04
DEVICECATEGORY05            DEVICECATEGORY06
DEVICECATEGORY07            DEVICECATEGORY08
DEVICECATEGORY09            DEVICECATEGORY10
DEVICECATEGORY11            DEVICECATEGORY12
DEVICECATEGORY13            DEVICECATEGORY14
DEVICECATEGORY15            DEVICECATEGORY16
DEVICECATEGORY17            DEVICECATEGORY18
DEVICECATEGORY19            DEVICECATEGORY20
INSTCOUNT                   INSTTYPE
HASDBOMRECORD               STARTMATCOUNT
STARTMATDEVIDS              STARTMATATTRIB
CONTINUEFROMDEVCOUNT        CONTINUEFROMDEVIDS
SUPPLIERCOUNT               SUPPLIERIDS
ALTDESTCOUNT                ALTDESTPROMSYSNAME
ALTDESTINVENTORYID          INSTSTAGENAME
INSTCOMMENT                 INSTDEVICEID
INSTPROID                   INSTINVENTORYID
INSTLEADTIME                INSTPROMSYSNAME
INSTMATTYPE                 INSTMATIDENT
INSTPROCSTATE               OUTPARTFLAG
OUTPARTCOUNT                OUTPARTBINCLASS
OUTPARTBINNAME              OUTPARTDEVNAME
OUTPARTPREDYIELD            OUTPARTTARGYIELD
PARAMETERIZATION            NENGPARAMS
NPLANPARAMS                 NTOTALPARAMS
PARAMNAME                   PARAMVAL
INSTLEAD
```

Figure 10: Sample structure definition menu.

of elements to retrieve from the array and for the PROMIS field which specifies exactly how many elements are in the array. Figure 10 shows the query for the DEVICE structure being defined.

DRIFS groups primitive structures by local database, allowing the user to retrieve all of the primitive structures defined for that local database. The user may also retrieve all structures with a particular slot name (e.g. "status" or "production-area"). The structures are displayed in the Interaction pane, and the user can select them to view their definitions in the Type-out pane. Note: retrieving primitive structures should not be confused with retrieving data from a local database. Primitive structure definitions are entirely within DRIFS.

5.1.4 Creating User Level Structures

User level structures combine slots from primitive structures, allowing data from separate databases to be integrated in a common data model. They are created using pop up menus that specify which primitive structures are to be used, and which slots to include from each primitive structure. One slot from each primitive structure must be identified as a key slot. DRIFS expects the data in key slots to be stored in the same format across primitive structures.

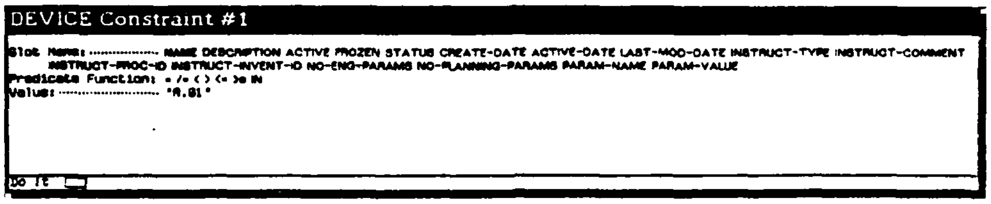
5.2 Retrieving Primitive Level Data

To initiate a DRIFS primitive level query, the user selects a primitive structure to retrieve and specifies the search criteria (known to DRIFS as query constraints) pertaining to that structure. The query constraints are entered via pop-up menus. Figure 11 shows how query constraints are specified using the DRIFS window interface.

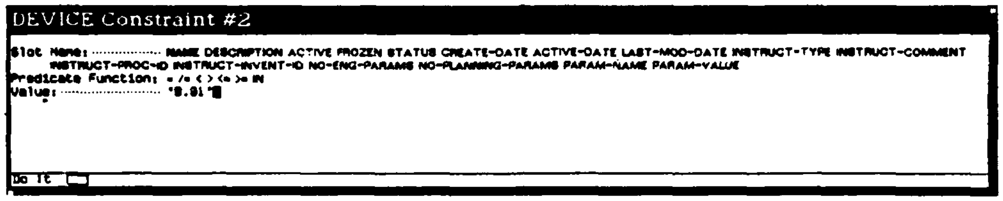
Once the query constraints have been specified by the user, DRIFS translates them into the query language of the local database and issue the proper query to the server for that database. Two methods were evaluated for communicating query commands to local database servers:

- Remote batch jobs.
- Ascii-translating character streams.

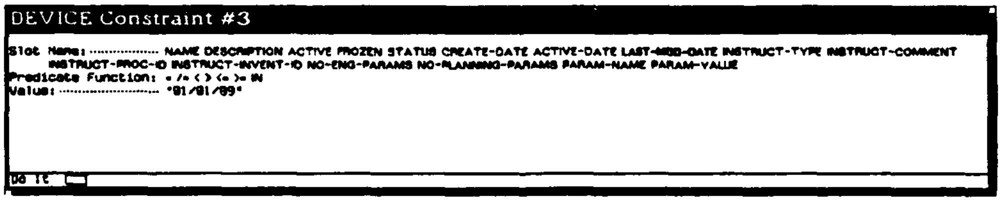
Batch jobs have the advantage of standard handshaking provided by the network protocols. Such handshaking allows for better handling of error conditions, and less programming overhead. However, response time for batch jobs can be unacceptably slow, depending on the configuration of the host computer. For example, EXGEFE (the host computer for PROMIS and the engineering test database at DLOG-II) gives batch jobs minimal priority at peek computing hours. In fact, during certain times of the day, batch processing virtually stops until the computing load is relieved. Another disadvantage to remote batch jobs is that they must initiate and release a new process each time a retrieval is requested. A large portion of the time it takes to run a PROMIS query is consumed in getting to the proper PROMIS menu and exiting the system. With batch jobs, this must be done each time a query is run.



```
Slot Name: ................. NAME DESCRIPTION ACTIVE FROZEN STATUS CREATE-DATE ACTIVE-DATE LAST-MOD-DATE INSTRUCT-TYPE INSTRUCT-COMMENT
      INSTRUCT-PROC-ID INSTRUCT-INVENT-ID NO-ENG-PARAMS NO-PLANNING-PARAMS PARAM-NAME PARAM-VALUE
Predicate Function: = /= < > <= >= IN
Value: ........................ "A.01"
```



```
Slot Name: ................. NAME DESCRIPTION ACTIVE FROZEN STATUS CREATE-DATE ACTIVE-DATE LAST-MOD-DATE INSTRUCT-TYPE INSTRUCT-COMMENT
      INSTRUCT-PROC-ID INSTRUCT-INVENT-ID NO-ENG-PARAMS NO-PLANNING-PARAMS PARAM-NAME PARAM-VALUE
Predicate Function: = /= < > <= >= IN
Value: ........................ "B.01"
```



```
Slot Name: ................. NAME DESCRIPTION ACTIVE FROZEN STATUS CREATE-DATE ACTIVE-DATE LAST-MOD-DATE INSTRUCT-TYPE INSTRUCT-COMMENT
      INSTRUCT-PROC-ID INSTRUCT-INVENT-ID NO-ENG-PARAMS NO-PLANNING-PARAMS PARAM-NAME PARAM-VALUE
Predicate Function: = /= < > <= >= IN
Value: ........................ "01/01/89"
```

Figure 11: Constraints used in a DEVICE primitive query.

These constraints request all DEVICE's with names beginning in "A" that have been modified after Jan. 1, 1989.

In contrast, an ascii-translating character stream is a direct connection to another computer via remote login. The stream has an output buffer through which a program can send commands to the remote host as if a user were typing them. The output from the host is piped into the stream's input buffer where it can be interpreted by the program. Ascii-translating character streams have the advantage of efficiency but are more prone to unanticipated errors. For example, if the host sends a system message to all users, the DRIFS will not know how to interpret that message, or, if a process times-out or aborts for any reason, DRIFS may continue to send commands to the non-existent process, not knowing it has been deactivated. Also, if a stream has been inactive for a certain length of time, the remote login session may terminate automatically in what is called an autologout.

Ascii-translating streams were used to communicate with the PROMIS and the engineering test databases on EXGEFE, and batch jobs were used to communicate with the financial database on TILDE. Batch jobs were acceptable for TILDE because even at high system load, there was little or no time between queuing and execution. LISP structures were defined for the PROMIS and engineering test databases to hold the character streams and the information necessary to generate them. Since direct network connections may not exist from the DRIFS computer to the local host, local database structures define a connection path which lists the hosts, or connection nodes, leading to the desired host. The ascii-translating character stream must login to each node separately until it reaches the desired host. The connection nodes hold the information necessary for login at each host. The retrieval connection is a LISP structure holding the character stream and its pertinent information. Local database structures were designed to allow for parallel query processing through use of more than one retrieval connection and a query queue and results queue. However, parallel processing was not implemented in the initial prototype, and these mechanism were not used. Figure 12 describes the LISP representation of the PROMIS local database and its connection nodes and retrieval connections.

5.2.1 Retrieving Primitive Structure Data from PROMIS

Note from Figure 5 that there is more than one network path between ESPRESSO and EXGEFE. To generate a retrieval connection, ESPRESSO can use DECnet

Description of PROMIS local database structure.

```
#<LOCAL-DB -64740176>
NAME:                PROMIS
HOST-NAME:           "exgefe"
SYSTEM-TYPE:         VMS
TELNET-PORT-NO:      23
CONN-PATH:           (#<CONN-NODE -75300401> #<CONN-NODE -75300157>)
LOCAL-DB:            PROMIS
INITIAL-NO-OF-RETRIEVAL-CONNS: 1
RETRIEVAL-CONNS:     (#<RETRIEVAL-CONN -75300152>)
QUERY-QUE:           SYS::|unbound|
RESULTS-QUE:         SYS::|unbound|

#<CONN-NODE -75300401>
HOST-NAME:           "al141"
SYSTEM-TYPE:         VMS
USERNAME:            "ruf"
PASSWORD:            "zimbabwe"

#<CONN-NODE -75300157>
HOST-NAME:           "exgefe"
SYSTEM-TYPE:         VMS
USERNAME:            "ruf"
PASSWORD:            "ayajaw"

#<RETRIEVAL-CONN -75300152>
OWNER:               #<LOCAL-DB -64740176>
LOCAL-DB:            PROMIS
STREAM:              #<IP::ASCII-TRANSLATING-CHARACTER-STREAM
-75300046>
IN-USE:              NIL
```

Figure 12: LISP representation of the PROMIS local database, connection nodes and retrieval connections.

expressly to connect to the local router, and then to EXGEFE, or it can use TCP/IP to connect to ALL41 and then login to EXGEFE through DECnet. Because the Explorer provides more comprehensive support for TCP/IP streams, the latter path was chosen. In order to specify this path, the connection nodes were defined as shown in Figure 12.

PROMIS can take as long as 3 minutes to activate the DataLink menu, which provides the general file extract commands. Exiting that menu and returning to VMS can require up to 2 minutes. To reduce overhead, the ascii-translating character stream loads the DataLink menu only once during initialization, and all queries are conducted without exiting DataLink.

PROMIS allows batch commands to be executed from any of its menus. The commands are stored in a *script file*, exactly as a user would type them in. When a script file is executed, PROMIS simply reads its command data from the file rather than the user's terminal. To minimize use of the character stream, DRIFS writes its queries to a script file on EXGEFE and uses the stream only to activate and monitor the progress of the script command. Due to a problem with the Explorer implementation of DECnet, files longer than about 600 bytes could not be transferred directly from the ESPRESSO to EXGEFE. The following work-around was used:

1. Script files are first copied from ESPRESSO to ALL41 using TCP/IP.
2. ESPRESSO issues a batch job on ALL41 to copy the script file to EXGEFE.
3. ESPRESSO reads EXGEFE's directory until the script file has been successfully copied from ALL41.
4. ESPRESSO deletes the script and batch files from ALL41.
5. Using the ascii-translating character stream, ESPRESSO issues a command to activate the script file from the DataLink menu.

Each script file includes commands to perform the specified query and to convert the PROMIS output file into DIF format. While the script is executing, DRIFS monitors its progress in the Type-out pane and watches for a key string at the end of the script file identifying its completion. Using DECnet, the DIF file is then read from EXGEFE and parsed into a format suitable for the primitive structure type being queried. A LISP flavor instance is created for each entry extracted from the PROMIS database, and these instances are listed in the Interaction pane. The user may then select them individually to view their contents in the Type-out pane.

5.2.2 Retrieving Primitive Structure Data from the Engineering Test Database

DRIFS uses a separate character stream connected to EXGEFE to query the engineering test database. This character stream communicates at the VMS command level, passing query constraints to the extract program (TDBEXTR) from the command line. Once the TDBEXTR command has been issued, DRIFS monitors its progress in the Interaction pane and watches for a VMS prompt indicating completion of the extract. Using DECnet, the a description file is then read to identify the format of the data file. Then, the data file is read and parsed into flavor instances of primitive structures. These are listed in the interaction pane.

TDBEXTR writes the output data file in wide tabular format. The width of the data file varies depending on the number of fields retrieved with each entry. Because of a networking problem, VMS files wider than 256 characters could not be transferred across the network. Therefore, DRIFS queries to the test database had to be restricted so as not to generate output files wider than 256 characters. A possible work-around for this networking problem is to write a program on EXGEFE to parse the tabular files into an acceptable format for network transfer. However, no work-around was implemented in the prototype.

5.2.3 Retrieving Primitive Structure Data from the Financial Database

To query data from the financial database, DRIFS creates a script file containing Ingres commands. The script file is copied to TILDE, and a batch file is then used to load Ingres and activate the script file. The script file instruct Ingres to retrieve the requested fields and copy them to an output file. When the script has completed, DRIFS reads the output file from TILDE, parsing the data into flavor instances of primitive structures, which are then listed in the interaction pane.

5.3 The User Level Interface

5.3.1 Combining PROMIS and Financial Data

To describe the DRIFS user level interface, an example user level structure, FIN-DEVICE, will be used. FIN-DEVICE combines data from two primitive level structures:

- DEVICE (from PROMIS)
- DEVICE-COST-DATA (from the financial database)

The primitive structure, DEVICE (Figure 13), is the template for retrieving device data from PROMIS, just as DEVICE-COST-DATA (Figure 14) is the template for retrieving cost data from the financial database. FIN-DEVICE (Figure 15) incorporates fields from DEVICE and DEVICE-COST-DATA which might be of interest to a production planner. Thus, retrieving FIN-DEVICE structures requires data to be integrated from PROMIS and the financial database.

DRIFS divides a user level query into separate primitive level queries and uses the key slots to match corresponding results from each query. For example, suppose the user wants to view all FIN-DEVICE's with device ID's beginning in "ASAM". He would query the FIN-DEVICE structure with that constraint. DRIFS would perform two separate primitive level queries. First it would retrieve device data from PROMIS by querying all DEVICE structures with NAME's beginning in "ASAM". Then it would retrieve cost data from the financial database by querying all DEVICE-COST-DATA structures with NAME's beginning in "ASAM". At this point, DRIFS compares the data in the key slots (in this case, the full NAME's) to match each DEVICE structure with its corresponding DEVICE-COST-DATA structure. Once these primitive structures are paired up, each pair is combined to form a FIN-DEVICE structure.

In the preceding example, the query constraints involved the key slots of the DEVICE and DEVICE-COST-DATA structures. If no key slot was included in the

DEVICE (PROMIS)

NAME:	string	SINGLETON
DESCRIPTION:	string	SINGLETON
ACTIVE:	string	SINGLETON
FROZEN:	number	SINGLETON
STATUS:	string	SINGLETON
CREATE-DATE:	string	SINGLETON
ACTIVE-DATE:	string	SINGLETON
LAST-MOD-DATE:	string	SINGLETON
INSTRUCT-TYPE:	string	LIST
INSTRUCT-COMMENT:	string	LIST
INSTRUCT-PROC-ID:	string	LIST
INSTRUCT-INVENT-ID:	string	LIST
NO-ENG-PARAMS:	number	SINGLETON
NO-PLANNING-PARAMS:	number	SINGLETON
PARAM-NAME:	string	LIST
PARAM-VALUE:	string	LIST

Figure 13: Description of DEVICE primitive level structure.

constraints, the user level query would be handled differently. For example, suppose the user wants to retrieve all FIN-DEVICE structures with current quarter yields less than 40%. In this case, DRIFS would still perform two primitive level queries. First it would retrieve all DEVICE-COST-DATA structures from the financial database with current quarter yields less than 40%. Then, it would use the data in the DEVICE-COST-DATA key slots (in this case, the NAME's) to build the query constraints for the DEVICE structure. DRIFS would use those query constraints to retrieve from PROMIS a matching DEVICE structure for each DEVICE-COST-DATA structure it has already retrieved from the financial database. The pairs are then combined to form FIN-DEVICE structures.

5.3.2 Combining PROMIS and Engineering Test Data

Integrating data from PROMIS and the engineering test database cannot be handled the manner described in Section 5.3.1, because there is not a one-to-one correspon-

```
DEVICE-COST-DATA (FINANCIAL-DB)
                NAME:  string              SINGLETON
      BARS-PER-SLICE:  number              SINGLETON
   CUR-QTR-FLOW-COST:  number              SINGLETON
       CUR-QTR-YIELD:  number              SINGLETON
  PREV-QTR-FLOW-COST:  number              SINGLETON
      PREV-QTR-YIELD:  number              SINGLETON
```

Figure 14: Description of DEVICE-COST-DATA primitive level structure.

```
FIN-DEVICE

Slots:
                NAME  (from DEVICE)
         DESCRIPTION  (from DEVICE)
              ACTIVE  (from DEVICE)
              FROZEN  (from DEVICE)
              STATUS  (from DEVICE)
         CREATE-DATE  (from DEVICE)
         ACTIVE-DATE  (from DEVICE)
       LAST-MOD-DATE  (from DEVICE)
      BARS-PER-SLICE  (from DEVICE-COST-DATA)
   CUR-QTR-FLOW-COST  (from DEVICE-COST-DATA)
       CUR-QTR-YIELD  (from DEVICE-COST-DATA)
  PREV-QTR-FLOW-COST  (from DEVICE-COST-DATA)
      PREV-QTR-YIELD  (from DEVICE-COST-DATA)

Key Slots:
                NAME  (from DEVICE)
                NAME  (from DEVICE-COST-DATA)
```

Figure 15: Description of FIN-DEVICE user level structure.

dence between PROMIS entries and entries in the test database: For each lot in PROMIS, there are several entries in the entries in the test database. For example, suppose the user wants to combine data from the following primitive structures:

- ACTIVE-LOT from PROMIS (Figure 16)
- LOT-TEST-RESULT from the engineering test database (Figure 17)

There are many LOT-TEST-RESULT's which correspond to each ACTIVE-LOT. To handle this situation, DRIFS allows user level structures to identify *sub-structures*. Sub-structures are primitive structures, such as LOT-TEST-RESULT, which have a many-to-one correspondence to other primitive structures, such as ACTIVE-LOT. The user level structure, ENG-ACTIVE-LOT (Figure 18), incorporates slots from ACTIVE-LOT which are of interest to an engineer. It also incorporates lot test data associated with each active lot by identifying LOT-TEST-RESULT as a sub-structure.

ACTIVE-LOT (PROMIS)

NAME:	string	SINGLETON
COMMENT:	string	SINGLETON
DEVICE-NAME:	string	SINGLETON
CUR-PROCESS-NAME:	string	SINGLETON
CUR-RECIPE-NAME:	string	SINGLETON
CUR-PROD-AREA-NAME:	string	SINGLETON
CUR-WORKCENTER-NAME:	string	SINGLETON
CUR-EQUIP-TYPE-NAME:	string	SINGLETON
CUR-EQUIP-UNIT-NAME:	string	SINGLETON
COMPLETION-CLASS:	string	SINGLETON
STATE:	string	SINGLETON
STATE-ENTRY-TIME:	string	SINGLETON
STAGE:	string	SINGLETON
TRACKING-STAGE:	string	SINGLETON
CUR-STEP-NUMBER:	number	SINGLETON
END-STEP-NUMBER:	number	SINGLETON
EMPL-ID-TRACKIN:	string	SINGLETON
EMPL-ID-TRACKOUT:	string	SINGLETON
STEP-COMMENT:	string	SINGLETON
QUEUE-TIME:	string	SINGLETON
STEP-START-DATE:	string	SINGLETON
STEP-END-DATE:	string	SINGLETON
RELEASE-TIME:	string	SINGLETON
STEP-START-SIZE:	number	SINGLETON
MECH-STEP-YIELD:	number	SINGLETON
NO-OF-DIE-IN:	number	SINGLETON
DIE-STEP-YIELD:	number	SINGLETON
CUR-MECH-LOT-YIELD:	number	SINGLETON
CUR-EFF-DIE-YIELD:	number	SINGLETON

Figure 16: Description of ACTIVE-LOT primitive level structure.

```
LOT-TEST-RESULT (TEST-DB)
            TESTER:  string                 SINGLETON
        TECHNOLOGY:  string                 SINGLETON
            DEVICE:  string                 SINGLETON
          LOT-NAME:  string                 SINGLETON
         TEST-DATE:  string                 SINGLETON
         TEST-TIME:  string                 SINGLETON
        TEST-TYPES:  string                 LIST
      TEST-RESULTS:  TEST-RESULT            LIST
```

Figure 17: Description of LOT-TEST-RESULT primitive level structure.

```
ENG-ACTIVE-LOT

Slots:
                 NAME  (from ACTIVE-LOT)
              COMMENT  (from ACTIVE-LOT)
          DEVICE-NAME  (from ACTIVE-LOT)
     CUR-PROCESS-NAME  (from ACTIVE-LOT)
      CUR-RECIPE-NAME  (from ACTIVE-LOT)
   CUR-PROD-AREA-NAME  (from ACTIVE-LOT)
  CUR-WORKCENTER-NAME  (from ACTIVE-LOT)
  CUR-EQUIP-TYPE-NAME  (from ACTIVE-LOT)
  CUR-EQUIP-UNIT-NAME  (from ACTIVE-LOT)
                STATE  (from ACTIVE-LOT)
                STAGE  (from ACTIVE-LOT)
      CUR-STEP-NUMBER  (from ACTIVE-LOT)
         STEP-COMMENT  (from ACTIVE-LOT)

Key Slots:
                 NAME  (from ACTIVE-LOT)

Sub-structures:
      LOT-TEST-RESULT  Key Slot: LOT-NAME
```

Figure 18: Description of ENG-ACTIVE-LOT user level structure.

6 Evaluations

6.1 Evaluation of DRIFS Prototype Implementation

The DRIFS prototype is successful in providing a standard data model and retrieval interface to data in three heterogeneous databases:

- **PROMIS Database.** The shop floor control system database used by DLOG-II, a Texas Instruments fabrication facility. This database resides on a VAX 8650 running VMS.
- **Engineering Test Database.** The database used by DLOG-II for storing IC probe and test results. This database resides on a VAX 8650 running VMS.
- **Financial Database.** A mock financial database created with Ingres. This database resides on a VAX 785 running ULTRIX.

6.1.1 Heterogeneous Hardware Environment

Because the financial database resides on a separate computer system from the DLOG-II VAX which stores the PROMIS and engineering test databases, the prototype demonstrates that the DRIFS concept can be implemented in a heterogeneous hardware environment.

6.1.2 DRIFS User Interface

The window-oriented menu interface of the DRIFS prototype provides a flexible and convenient method for defining representations of the local fabrication databases. The extensive text provided by the PROMIS help feature is a useful aid in designing primitive level structures.

6.1.3 DRIFS Data Model

The model for DRIFS primitive structures is sufficient for defining representations of each of the local fabrication databases. In addition to providing homogeneity, the primitive structures are easily integrated at the user level.

The use of key slots and substructures allows primitive data from separate fabrication databases to be integrated via user level structures. Additionally, user level structures provide a means of defining specialized "views" of the fabrication data for certain groups (i.e. engineers, production planners, etc.) This was demonstrated in the prototype through the ENG-ACTIVE-LOT and FIN-DEVICE user level structures, tailored for engineers and production planners, respectively.

6.1.4 Access Times

While the Explorer LISP machine used to implement DRIFS provides an excellent environment for rapid prototyping, networking from the Explorer to other computers is slow and cumbersome. The work-around to the networking problem described in Section 5.2.1 adds substantial overhead to running PROMIS database queries from DRIFS. This overhead, combined with the already slow transfer rate (about 500 to 800 bytes/sec) between the Explorer and the DLOG-II VAX severely limits the performance of PROMIS queries from DRIFS. Table 1 shows the access times required to retrieve one DEVICE structure from PROMIS. Note that transferring the PROMIS script file to EXGEFE accounts for 68% of the total access time (due to the networking work-around). Since transfer time for script files is constant with respect to the number of DEVICE's retrieved from PROMIS, this percentage become less significant (15%) when a larger number (20) of DEVICE structures are retrieved (see Table 2).

Access times for queries to the engineering test and financial database are more acceptable. Table 3 shows the access times for retrieving test data for an active lot with 15 LOT-TEST-RESULT entries. Table 4 shows the access times for retrieving test data for an active lot with 114 LOT-TEST-RESULT entries. Notice that at 114 entries, reading the output file over the network is the dominant time factor. Table 5 shows the access times for retrieving the cost data associated with one

Access Times to Retrieve one DEVICE Structure from PROMIS				
	Transfer Script File to EXGEFE	Execute Script File on EXGEFE	Read DIF Output File from EXGEFE	Total Access Time
Trial 1	0:36	0:11	0:11	0:58
Trial 2	0:33	0:07	0:10	0:50
Trial 3	0:34	0:09	0:07	0:50
Trial 4	0:35	0:08	0:07	0:50
Average	**0:35**	**0:09**	**0:09**	**0:52**
Average %	**68%**	**17%**	**17%**	

Table 1: Access times for one DEVICE structure.

Times are in M:SS format.

Access Times to Retrieve 20 DEVICE Structures from PROMIS				
	Transfer Script File to EXGEFE	Execute Script File on EXGEFE	Read DIF Output File from EXGEFE	Total Access Time
Trial 1	0:38	1:46	1:55	4:19
Trial 2	0:41	2:19	2:00	5:00
Trial 3	0:40	1:46	1:59	4:25
Trial 4	0:39	1:49	1:58	4:26
Average	**0:40**	**1:55**	**1:58**	**4:33**
Average %	**15%**	**42%**	**43%**	

Table 2: Access times for 20 DEVICE structures.

Times are in M:SS format.

Access Times to Retrieve 15 LOT-TEST-RESULT Structures from the Test Database			
	Run TDBEXTR Extract on EXGEFE	Read Output File from EXGEFE	Total Access Time
Trial 1	0:19	0:07	0:26
Trial 2	0:17	0:10	0:27
Trial 3	0:20	0:08	0:28
Trial 4	0:22	0:08	0:30
Average	**0:20**	**0:08**	**0:28**
Average %	**71%**	**29%**	

Table 3: Access times for 15 LOT-TEST-RESULT structures.

Times are in M:SS format.

device. Table 6 shows the access times for retrieving the cost data associated with 10 devices. The Ingres execution time increases from 21 sec. to 32 sec. (an increase of 52%) when the number of DEVICE-COST-DATA structures retrieved increases from one to 10 (an increase of 990%). These figures indicate that with small queries, a major portion of the access time is overhead in loading and executing the Ingres script file.

6.2 Evaluation of the DRIFS Concept

While DRIFS provides a standard retrieval interface and data model for heterogeneous fabrication databases, its effectiveness can be limited by the existing interfaces to the individual local databases and by networking demands. For example, the only interface to PROMIS data is through menus intended to handle interactive commands from a user console. Automated interaction with these menus is a cumbersome and error-prone method. DRIFS could be more effective if each local database provided a retrieval interface designed to handle automated data retrieval. Even without such interfaces, however, DRIFS can provide an excellent means for

Access Times to Retrieve 114 LOT-TEST-RESULT Structures from the Test Database			
	Run TDBEXTR Extract on EXGEFE	Read Output File from EXGEFE	Total Access Time
Trial 1	0:31	0:25	0:56
Trial 2	0:27	0:27	0:54
Trial 3	0:23	0:27	0:50
Trial 4	0:22	0:33	0:55
Average	**0:26**	**0:28**	**0:54**
Average %	**48%**	**52%**	

Table 4: Access times for 114 LOT-TEST-RESULT structures.

Times are in M:SS format.

Access Times to Retrieve one DEVICE-COST-DATA Structure The Financial Database				
	Transfer Script File to TILDE	Execute Script File on TILDE	Read Ingres Output File from TILDE	Total Access Time
Trial 1	0:04	0:24	0:08	0:36
Trial 2	0:04	0:20	0:12	0:36
Trial 3	0:07	0:21	0:13	0:41
Trial 4	0:04	0:19	0:12	0:35
Average	**0:05**	**.0:21**	**0:11**	**0:37**
Average %	**13%**	**57%**	**30%**	

Table 5: Access times for one DEVICE-COST-DATA structure.

Times are in M:SS format.

Access Times to Retrieve 10 DEVICE-COST-DATA Structures The Financial Database				
	Transfer Script File to TILDE	Execute Script File on TILDE	Read Ingres Output File from TILDE	Total Access Time
Trial 1	0:05	0:32	0:12	0:49
Trial 2	0:04	0:41	0:12	0:55
Trial 3	0:04	0:31	0:12	0:47
Trial 4	0:04	0:25	0:11	0:40
Average	**0:04**	**0:32**	**0:12**	**0:48**
Average %	**08%**	**67%**	**25%**	

Table 6: Access times for 10 DEVICE-COST-DATA structures.

Times are in M:SS format.

off line data retrieval in converting to a homogeneous fabrication database.

References

[1] Dolins, S.B., A. Srivastava, I. Mani, and A. Myjack. *Intelligent Factory Management System*. Technical Report, AI Laboratory, Texas Instruments, Dallas, TX. 1988.

[2] Ephraim, O. M. *Integrating CAD/CAM Systems*. Conference on Computer Aided Manufacturing and Productivity. London, 1981.

[3] *Explorer Technical Summary*, Rev. C, Texas Instruments, Inc., Austin, TX. 1987.

[4] *GEFE Engineering Test Database (TDB) User's Guide Document*. Texas Instruments, 1988.

[5] Heytens, M. *Database Schema of the MIT CAFE System*. Massachusetts Institute of Technology, 1987.

[6] Heytens, M. and R. Nikhil. *GESTALT: An Expressive Database Programming System*. Massachusetts Institute of Technology, 1987.

[7] Heytens, M. *GESTALT User's Manual*. Massachusetts Institute of Technology, 1987.

[8] *INGRES Reference Manual*. Version 3.0, VAX/VMS, Relational Technology, Inc., Berkeley, CA. 1984.

[9] Katz, R. H. "Database Management and Computer-Assisted VLSI Fabrication." *Database Engineering*. Vol. 7. No. 2, pp. 35-38. IEEE Computer Society, 1984.

[10] Landers, T. and R. Rosenberg. *An Overview of Multibase*. North-Holland Publishing Co., 1982.

[11] Moore, R. D. *CIM Applications: An Architectural Overview*. Harris Semiconductor.

[12] *PROMIS Standard System Guide*. Vol. 1-2, Rel. 4.2, PROMIS Systems Corporation, 1987.

[13] Ruf, M. *DRIFS: Data Retrieval Interface for Fabrication Systems.* Third DARPA/SRC Workshop on Computer-Integrated Manufacturing of Integrated Circuits, Stanford, CA. 1988.

[14] Smith, J.M. et. al. *Multibase - integrating heterogeneous distributed database systems.* Conference Proceedings of National Computer Conference. Chicago, 1981.